

Copy of the related co-pending application and drawings
US Serial Number 11/157,113, filed 06/20/2005,
First Named Inventor – Alexandre V. Grigorovitch.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Multi-Thread Multimedia Processing

Inventors:

Alexandre V. Grigorovitch,

Gaurav Lochan,

and

Patrick N. Nelson

ATTORNEY'S DOCKET NO. MS1-2564US

BACKGROUND

Rendering multimedia can be a processor-intensive exercise. Insufficient processing power can result in glitches in media rendering, like a delay in a soundtrack or video playback of a movie.

In part to address this need for greater processing power, computing devices have been designed to execute more than one processing thread at a time. Some media-rendering processes have components that take advantage of these devices by allocating and using their own threads.

But building components able to allocate their own threads can be more difficult than building components without this ability. Also, these components may use significant processing resources to create and manage their own threads.

These components also may allocate threads poorly. They may use too many or too few processing resources, often because they are not fully aware of upstream and downstream components of a multimedia pipeline of which they are a part. They may also create too many threads, thereby wasting processing resources used to switch between threads.

SUMMARY

This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify essential features or determine the scope of the claimed subject matter.

Systems and/or methods (“tools”) are described below that enable multi-threaded multimedia processing. The tools may allocate threads for components of a multimedia pipeline based on input/output characteristics of the components.

1 The tools may also allocate threads through a controller instead of through
2 components, thereby potentially reducing the time and complexity needed to build
3 components of a multimedia pipeline.

4 In some embodiments, the tools allocate threads and priorities for those
5 threads based on the relative importance given end components of a multimedia
6 pipeline. By so doing a user may enjoy higher-quality rendering on media that is
7 important to the user.

8 9 **BRIEF DESCRIPTION OF THE DRAWINGS**

10 Fig. 1 illustrates an exemplary operating environment in which various
11 embodiments can operate.

12 Fig. 2 illustrates an exemplary media pipeline topology.

13 Fig. 3 is an exemplary flow diagram for allocating threads and/or assigning
14 priorities to components of a media pipeline.

15 Fig. 4 illustrates inputs and two allocated threads for the pipeline of Fig. 2.

16 Fig. 5 illustrates four allocated threads for the pipeline of Fig. 2.

17 Fig. 6 is an exemplary process for allocating threads and/or assigning
18 priorities to components of multiple media pipelines.

19 Fig. 7 illustrates two exemplary media pipeline topologies.

20 Fig. 8 illustrates allocated threads for the exemplary pipelines of Fig. 7.

21 Fig. 9 illustrates an exemplary media pipeline topology having composition
22 components.

23 Fig. 10 is an exemplary process for allocating threads for a multimedia
24 pipeline having a composition component.

1 The same numbers are used throughout the disclosure and figures to
2 reference like components and features.

3 4 **DETAILED DESCRIPTION**

5 *Overview*

6 The following document describes system(s) and/or method(s) (“tools”) for
7 managing a multimedia pipeline. The tools may allocate threads and/or priorities
8 for components of a multimedia pipeline.

9 In one embodiment, for instance, the tools allocate one thread to a
10 particular component or series of components and another thread to another
11 component or series of components. If there is a potential conflict by one thread
12 taking too many processing resources such that it may cause a glitch or delay, the
13 tools can set a priority for each thread so that a glitch or delay happens on a less
14 important thread.

15 The tools may also, in another embodiment, allocate and control threads
16 through a controller. One advantage of a controller doing so rather than
17 components of a multimedia pipeline, is that components often know less about
18 priorities and likely usage of upstream and downstream components than the
19 controller. Using a controller the tools may enable more efficient processor usage.
20 The tools may also enable components of media applications to more easily be
21 developed by reducing the complexity of programming for these components.

22 23 *Exemplary Operating Environment*

24 Before describing the tools in detail, the following discussion of an
25 exemplary operating environment is provided to assist the reader in understanding

1 where and how the tools may be employed. The description provided below
2 constitutes but one example and is not intended to limit application of the tools to
3 any one particular operating environment.

4 Figure 1 illustrates one such operating environment generally at 100
5 comprising a computer 102 having processors 104 and computer-readable media
6 106. The processors are capable of accessing and/or executing the computer-
7 readable media. The computer-readable media comprises or has access to a media
8 application 108, which comprises components capable of acting in a multimedia
9 pipeline. These components include: an audio input 110; an audio decoder 112; an
10 audio Sample Rate Converter (SRC) 114; a tee 116; a Fast Fourier Transform
11 (FFT) 118; an audio renderer 120; a visualization sink (VizSink) 122; a video
12 input 124; a video decoder 126; and a video renderer 128. These components are
13 capable of transforming media data, performing effects on or using media data,
14 and/or duplicating media data for use by multiple components. Other types of
15 components can also be used, like those capable of composing media data from
16 two sources and passing it to another component, though these are not illustrated
17 in this environment.

18 The computer-readable media also comprises or has access to a controller
19 130. The controller is capable of managing a media pipeline, such as by allocating
20 threads and/or priorities to components of the media pipeline.

21 22 *Exemplary Multimedia-Pipeline Topology*

23 The following exemplary topology for a multimedia pipeline is described to
24 aid the reader in understanding one way in which a particular pipeline can be
25 oriented. The multimedia pipeline is represented as a topology of components,

1 such as transformation and effect components. This pipeline is not intended to
2 limit the application of the tools to multimedia applications, multimedia pipelines,
3 or this particular pipeline.

4 Figure 2 illustrates an exemplary topology 200 for a multimedia pipeline
5 having the components shown in media application 108. This topology shows the
6 orientation of the components of the media application and their interaction with
7 each other. Audio input 110 and video input 124 pass audio and video data to
8 other components. These components are called transform components because
9 they pass data to a single other component of the topology. A group of transform
10 components can act in series. Audio input 110 passes data to audio decoder 112,
11 which transforms (here decodes) this input and then passes it to SRC 114. The
12 SRC transforms the decoded data and passes the results to tee 116. Audio decoder
13 and SRC are also transforms that act in series. Likewise, video decoder 126
14 transforms data from video input 124 and passes the transformed data to video
15 renderer 128. Video decoder 126 is also a transform that acts in series.

16 Tee 116, however, acts to send data to two different components, here FFT
17 118 and audio renderer 120. The tee operates in the topology to permit data to
18 branch off into different pipes of the pipeline. FFT 118 acts as a transform,
19 sending its output to VizSink 122.

20 This topology shows how multimedia data (here audio and video data) can
21 be rendered and stored. This pipeline acts to render video on a computer display,
22 render audio through speakers, and store audio. The computer 102 can present to
23 a user a music video, movie, or other program by the media application
24 performing this multimedia pipeline.
25

Allocating Threads and Assigning Priorities

The following discussion describes exemplary ways in which the tools allocate threads and/or assign priorities to components of a multimedia pipeline.

Referring to Figure 3, an exemplary flow diagram 300 for allocating threads and/or assigning priorities is shown. Flow diagram 300 is illustrated as a set of actions by, and accompanying communications between, elements of environment 100, though these actions are not limited to the elements of environment 100. The actions and accompanying communications are marked with arrows. This flow diagram may be implemented in any suitable hardware, software, firmware, or combination thereof. In the case of software and firmware, this diagram represents sets of operations implemented as computer-executable instructions.

Flow diagram 300 is described generally and with illustrated examples. In one example, the tools allocate threads based on non-overlapping inputs to components in a pipeline's topology. In another example, the tools assign threads having different priorities based on components having non-overlapping inputs in a pipeline's topology and relative importance of particular components.

Arrow 1 receives topology and/or priority information for components of a multimedia pipeline. Arrow 1 may receive topology information comprising a topology mapping input and output characteristics of components of a pipeline or instead information about components of the pipeline from which arrow 1 may ascertain the topology. In some cases a media application is capable of sending a complete topology. In some others, each component can send information about how it interacts with neighboring components.

1 In the first illustrated example, controller 130 receives topology-related
2 information from each of the components of media application 108 setting out
3 each of their input/output characteristics. Controller 130 receives the following
4 information:

5
6 audio input 110 outputs to audio decoder 112;

7 audio decoder 112 receives from audio input 110 and outputs
8 to SRC 114;

9 SRC 114 receives from audio decoder 112 and outputs to tee
10 116;

11 tee 116 receives from SRC 114 and outputs to both FFT 118
12 and audio renderer 120;

13 FFT 118 receives from tee 116 and outputs to VizSink 122;

14 VizSink receives from FFT 118 and does not output;

15 audio render 120 receives from tee 116 and does not output;

16 video input 124 outputs data to video decoder 126;

17 video decoder 126 receives from video input 124 and outputs
18 to video renderer 128; and

19 video renderer 128 receives from video decoder 126 and does
20 not output.

21
22 Each of these inputs/outputs are illustrated in Figure 4 and represented by
23 “I_n”, where “_n” represents an input/output. Based on this input/output information,
24 the controller ascertains whether or not there are any overlapping inputs to end
25 components of the pipeline. Thus, the controller ascertains the direct and indirect

1 inputs to each end component: video renderer 128; VizSink 122; and audio
2 renderer 120. The audio renderer has the following inputs: I₁; I₂; I₃; and I₄. The
3 VizSink has the following inputs: I₁; I₂; I₃; I₄; and I₅. The video renderer has the
4 following inputs: I₆ and I₇. Based on this information, the controller can ascertain
5 what topology is being used. This may include ascertaining which components
6 provide a direct or indirect input to each of the end components and whether each
7 end component has an overlapping input within another end component. Here the
8 controller ascertains that video renderer 128 has no overlapping inputs with
9 VizSink 122 or audio renderer 120. It also ascertains that VizSink 122 and audio
10 renderer 120 having overlapping inputs (I₁, I₂, I₃, and I₄).

11 Arrow 2 allocates threads and/or priorities to components of a multimedia
12 pipeline. In the first illustrated example shown in Figure 4, controller 130
13 received topology-related information and ascertained the topology for the
14 pipeline. The controller also ascertained which components go to end components
15 have an overlapping input. The controller allocates threads based on this topology,
16 in this case a single thread to end components having an overlapping input and a
17 single thread to each end component not having an overlapping input. Here the
18 controller allocates a thread to each non-overlapping group of components of
19 topology 200 of Figure 2. The set of audio input 110 to audio renderer 120 and
20 VizSink 122 is one group because their end components have overlapping inputs.
21 Video input 124 to video renderer 128 are another group. The separate threads are
22 shown in boxes and marked Thread 1 and Thread 2 in Figure 4.

23 By so allocating, each thread does not overlap, permitting each thread to be
24 processed by separate cores or processors. This may enable a computer capable of
25

1 executing multiple threads at once to perform multiple pipes of a multimedia
2 pipeline also at once.

3 In a second illustrated embodiment, the tools allocate priorities to threads
4 and/or allocate additional threads based on priority information received. For the
5 purposes of this illustration, the same topology 200 is received or ascertained,
6 shown in Figure 2.

7 Arrow 1 receives information indicting that audio renderer 120 has a higher
8 priority than VizSink 122 and video renderer 128. With this information,
9 controller 130 can allocate additional threads and/or assign priorities to threads.

10 First, arrow 1 can ascertain, based on the topology information, which
11 components provide an input to a high-priority component. Arrow 1 can assign a
12 high priority to each thread having these components. Here controller 130
13 ascertains that audio input 110, audio decoder 112, SRC 114, and Tee 116 are
14 components providing input to the high-priority component audio renderer 120.
15 The controller may assign a high priority to thread 1 of Figure 4 based on this
16 determination. The controller may also assign a low priority to thread 2 of Figure
17 4 based on it not providing input to a high-priority component.

18 Second, arrow 1 can ascertain if a component not providing an input to the
19 high-priority component may conflict with any of these components. Here the
20 controller may ascertain, in topology 200, that audio renderer 120 of thread 1 may
21 conflict with FFT 118 or VizSink 122. This conflict can cause a delay or glitch in
22 audio renderer 120 and/or VizSink 122. To address this, controller 130 may set a
23 high priority for audio renderer 120 and a low or lower priority for VizSink 122
24 and FFT 118 so that a glitch or delay will first go to VizSink 122 and FFT 118
25 rather than audio renderer 120. To do so, arrow 2 allocates two addition threads to

1 thread 1 of Figure 4—threads 3 and 4. It also assigns a high priority to thread 4
2 and a low priority to thread 3. These new threads and reduction to thread 1 are
3 shown in boxes in Figure 5.

4 Controller 130 allocates these threads and sets the priority of thread 4 above
5 that of thread 3 and thread 1 above that of thread 2. It sets thread 1 above that of
6 thread 2 because the high-priority component of thread 4 receives inputs from
7 thread 1. In this way, a lack of processing resources will cause a glitch or delay
8 first to thread 3 or thread 2 rather than thread 4 or thread 1. Processors 104
9 execute these threads based on their priority. If thread 2 conflicts with thread 1
10 (which feeds to thread 4) or thread 4, thread 1 or 4 will be executed at higher
11 priority and thread 2 will be executed (or not executed) at a lower priority. This
12 may cause thread 2 to be delayed or allowed to glitch. Likewise, if thread 3
13 conflicts with thread 4, thread 4 will be executed at higher priority. This may
14 cause thread 3 to be delayed or allowed to glitch.

15 Having each component allocate and control a thread often requires
16 complex programming for each component. And each component may not have
17 sufficient information to properly allocate threads or may allocate too many
18 threads. Unnecessary threads can waste computational resources. A controller,
19 however, may require significantly less processor usage for allocating and
20 controlling threads and/or may allocate fewer threads. Also, the controller can
21 enable components to not need to be programmed with this ability, potentially
22 saving programming time and effort. The controller may also better allocate and
23 control the multimedia pipeline of which the components are a part than the
24 components themselves.

Multiple Media Pipelines

The following discussion describes exemplary ways in which the tools allocate threads and/or assign priorities to components of multiple media pipelines. In some situations a media application can perform multiple media pipelines, such as by showing a music video and an audio/video news clip. Components of these pipelines may conflict with each other and with components of another pipeline, potentially causing delays or glitches.

In Figure 6, an exemplary process 600 is shown illustrated as a series of steps representing individual operations or acts performed by elements of the operating environment 100 of Figure 1, though this process is not limited to environment 100. This and other processes disclosed herein may be implemented in any suitable hardware, software, firmware, or combination thereof; in the case of software and firmware, these processes represent a set of operations implemented as computer-executable instructions stored in computer-readable media 106 and executable by processors 104.

Step 602 instructs components of a media application to not open threads. In some cases, components can open their own threads. Opening threads, as well as other management and control, may instead be maintained by the tools.

Step 602 also requests information from the media application and/or other sources. This information permits intelligent control by the tools. Step 602 can request input/output information for each component of each pipeline, priority information for each component, information regarding in what way the media of each pipeline is to be used or viewed, and information regarding the value of the media of each pipeline. Some of this information may be requested from other sources, such as a source indicating that media of one pipeline is copyright

1 protected, a purchase history indicating that it was received free or was purchased
2 from a particular website, and the like.

3 Step 604 receives these requests and gathers information. Step 604 also
4 acts to not open threads. The gathered information may comprise one or more of
5 the following: input/output information indicating input/output characteristics of
6 components; priority information regarding one or more components; usage
7 information indicating how media of a pipeline is to be used or viewed; and value
8 information indicating a value of media of a pipeline.

9 In an illustrated embodiment, media application 108 gathers information
10 indicating that two media pipelines may be generated, that each is for audio/visual
11 media, and each has a topology similar to that of topology 200 of Figure 2. These
12 two media pipelines are shown for illustration in Figure 7 and marked 700a and
13 700b.

14 Media application also gathers information indicating that VizSink 122a
15 and VizSink 122b are high priority, and that audio renderer 120a, audio renderer
16 120b, video renderer 128a, and video renderer 128b are lower in priority.

17 Step 604 can also gather information regarding in what way the media of
18 each pipeline is to be used or viewed and information regarding the value of the
19 media of each pipeline. Here media application gathers value information
20 indicating that the media of topology 700a is for a free news clip from an Internet
21 website and that the media of topology 700b is for a high-definition music video
22 with copyright protection that was purchased from another Internet website. This
23 copyright and purchasing information may also be gathered by other sources. The
24 media application gathers usage information indicating that the music video of
25 topology 700b is to be rendered in a pane larger than that of the news clip and also

1 in the front ground relative to the new clip of topology 700a. Step 606 sends the
2 gathered information to the tools, which is received by step 608.

3 Step 610 ascertains topologies based on the information. Step 610 may do
4 so as set forth above in flow diagram 300 based on input/output information. Step
5 610 can also receive a built topology from a media application.

6 Step 612 allocates threads based at least in part on the topology and/or
7 priority information received. The tools can act according to flow diagram 300 to
8 provide, for example, four threads for topologies 700a and 700b (each set of
9 threads similar to those shown in Figure 5). These four threads for each of
10 topologies 700a and 700b are shown in Figure 8.

11 Step 614 assigns priority to components based on information indicating
12 the importance or value of the different medias. For example, controller 130 can
13 ascertain which media of which topology is of higher importance. In this case, the
14 controller can ascertain that the music video of topology 700b is more important
15 because it is in the foreground relative to the news clip of topology 700a. It can
16 also make this determination based on the music video be in high definition,
17 having copyright protection, or having been paid for, each of which is not true of
18 the news clip. Thus, based on any one or multiple value indicators, the tools may
19 allocate priority to one media pipeline over another.

20 Based on this higher value of topology 700b, controller 130 assigns a
21 highest priority to threads 5 and 7, a second highest priority to threads 6 and 8, a
22 third highest priority to threads 1 and 3, and a fourth highest priority to threads 2
23 and 4. Each of these threads and their priorities is shown in Figure 8. By so
24 doing, the controller allocates threads and priorities so that VizSink 122b and the
25 components from which it receives inputs will be the last to be delayed or have a

1 glitch, followed by audio renderer 120b and video renderer 128b. After this, the
2 VizSink 122a and its inputs will not be delayed or a glitch caused prior to audio
3 renderer 120a or video renderer 128a being delayed or having a glitch.

4 Step 616 executes the components according to their threads and thread's
5 priority. Here processors 104 execute the components following the threads.
6 Threads 1, 2, 5, and 6 can be processed first, though thread 5 will have highest
7 priority. Threads 3 and 4 will follow thread 1. Threads 7 and 8 will follow thread
8 5. In some scenarios many different threads may be executed at once (such as
9 with multi-core or multiple processor machines). Assuming three processors, for
10 instance, thread 5 may execute on a first processor, thread 6 on a second
11 processor, and threads 1 and 2 (not simultaneously) on a third processor. Threads
12 3 and 4 may execute on the third processor or, possibly thread 8 may execute on
13 the third processor if thread 6 is still executing on the second processor and thread
14 7 is executing on the first processor after the completion of thread 1.

15 As is evident, many different threads may need to be executed; allocating
16 which gets highest priority enables important components to be executed at a
17 higher quality. Allocating threads may also reduce overall processor load by
18 reducing a number of threads and costly switching between threads, such as when
19 each component opens its own thread.

20 In terms of a user's perspective, the user will generally see his or her music
21 video at a higher quality than the news clip. The user probably prefers this, as he
22 or she paid for it, chose to put it in the foreground, and because it is of a kind of
23 media where quality is more important (being in high definition).

1 *Allocating Threads Based on Time/Resources Needed*

2 As set forth above, the tools may allocate threads and priorities to decrease
3 processor usage and improve quality of particular media components. The tools
4 may also allocate threads to complete a component of a multimedia pipeline more
5 quickly and/or with fewer resources.

6 Assume, for example, that the tools receive a multimedia topology having
7 one or more composition components, such as topology 900 of Figure 900.
8 Composition components receive input from two or more other components
9 capable of being processed in parallel.

10 In topology 900, for instance, video inputs 124c, 124d, and 124e pass data
11 to video decoders 126c, 126d, and 126e, respectively. Two of these video
12 decoders pass their data to composition 902, which then passes it to function 904.
13 The third decoder 126e passes its data to composition 906, which also receives
14 input from function 904.

15 Following the flow diagrams and/or processes set forth above, the tools can
16 ascertain or receive input/output information or this topology according to step
17 1002 of process 1000 of Figure 10. Process 1000 is shown illustrated as a series
18 of steps representing individual operations or acts, some of which are described
19 performed by elements of the operating environment 100 of Figure 1 and some of
20 which are not. Process 1000 is described using environment 100 to aid the reader
21 in understanding ways in which the process may be implemented but is not
22 intended to limit the scope of the process to this environment.

23 Step 1004 receives timing information, such as information indicating
24 likely processing usage for components of a multimedia pipeline. The tools can
25

1 receive information from a media application like 108 of Figure 1, indicating that
2 the likely processing time or usage of the video inputs and decoders, for instance.

3 Step 1006 ascertains which is faster or requires fewer resources: performing
4 each of the parallel components and their input components in parallel with
5 separate threads or in series with a single thread. Step 1006 can ascertain the
6 parallel components inputting to the composition component and those
7 components inputting the parallel components as set forth in step 1002, flow
8 diagram 300, or process 600. Step 1006 may ascertain which option is faster
9 based on the timing information and a time to switch between threads.

10 Controller 130 can ascertain, for instance, that a thread having video input
11 124c, video decoder 126c, video input 124d, and video decoder 124d takes less
12 time or processing usage than two separate threads having just video input
13 124c/video decoder 126c and just video input 124d/video decoder 126d. This is
14 possible because of the amount of processing needed by a computing device to
15 switch between threads. Here the time to switch can be more than is used by
16 either video input 124c/video decoder 126c or video input 124d/video decoder
17 126d.

18 Step 1008 allocates separate threads for the parallel components and their
19 input components if performing separate threads is faster or requires fewer
20 processing resources than performing one thread.

21 Assume, for example, that five threads may be allocated for topology 900
22 of Figure 9 (labeled threads 1-5). Controller 130 can receive information
23 indicating that each of threads 1, 2, and 3 will take about the same amount of time
24 and that thread 4 will take longer than threads 1, 2, or 3. The controller can then
25 allocate thread 1 and 2 to perform first on two processors (or cores of a multi-core

processor) and, once 1 and 2 are complete, allocate that thread 3 run on one of the processors while thread 4 is allocated to run on the other. In this manner, thread 5 may complete quickly and without needing to spend processor time switching between too many threads.

If, on the other hand, the controller receives information indicating that thread 3 will take a small amount of time or processing resources, it may be more efficient to finish thread 5 by allocating a single thread (not shown) to the current threads 1 and 3 or 2 and 3. This is because the time and resources to switch threads is less than that for executing thread 3.

In this and similar ways the tools may allocate threads and assign priorities. The tools may operate through a central controller capable of determining these factors, which can reduce programming time needed to develop media applications and components. The tools may also save processing time and improve quality of multimedia pipelines and other pipeline-oriented processes.

Conclusion

The above-described systems and methods enable multi-thread multimedia processing. This multi-threaded multimedia processing can enable efficient, fast, and high quality media rendering. Although the system and method has been described in language specific to structural features and/or methodological acts, it is to be understood that the system and method defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed system and method.

CLAIMS

1
2
3 **1.** One or more computer-readable media having computer-readable
4 instructions therein that, when executed by a computer, cause the computer to
5 perform acts comprising:

6 receiving topology information indicating input/output characteristics of
7 components of a multimedia pipeline; and

8 allocating multiple threads for the components of the multimedia pipeline
9 based on the topology information.

10
11 **2.** The media of claim 1, wherein the topology information comprises a
12 topology mapping input/output characteristics of all of the components.

13
14 **3.** The media of claim 1, wherein the topology information comprises
15 input/output characteristics of each of the components, further comprising
16 ascertaining a topology based on the topology information, and wherein the act of
17 allocating is based on the topology.

18
19 **4.** The media of claim 3, wherein the topology information is received
20 from each of the components of the multimedia pipeline.

1 **5.** The media of claim 1, wherein the multimedia pipeline comprises
2 two or more end components and further comprising:

3 ascertaining which end components have an overlapping input with that of
4 another of the end components to provide one or more sets of overlapping-input
5 end components and one or more non-overlapping-input end components, and

6 wherein the act of allocating comprises:

7 allocating a thread to each set of the overlapping-input end
8 components and all components providing a direct or indirect input to any
9 of the overlapping-input end components of that set; and

10 allocating a thread to each of the non-overlapping-input end
11 components and all components providing a direct or indirect input to each
12 of the non-overlapping input end components.

13
14 **6.** The media of claim 1, further comprising receiving priority
15 information indicating a priority for one of the components to provide a high-
16 priority component and one or more low-priority components and wherein the act
17 of allocating comprises:

18 ascertaining which components provide inputs to the high-priority
19 component; and

20 assigning a high priority to each thread having one or more of the
21 components that provide inputs to the high-priority component.

1 7. The media of claim 1, further comprising receiving priority
2 information indicating a priority for one of the components to provide a high-
3 priority component and one or more low-priority components and wherein the act
4 of allocating comprises:

5 ascertaining whether one or more components conflict with the high-
6 priority component or any components providing input to the high-priority
7 component; and

8 allocating a low-priority thread to the one or more conflicting components.
9

10 8. One or more computer-readable media having computer-readable
11 instructions therein that, when executed by a computer, cause the computer to
12 perform acts comprising:

13 receiving input/output information and priority information regarding
14 input/output characteristics and priorities for first components of a first multimedia
15 pipeline and second components of a second multimedia pipeline; and

16 allocating, based on the input/output information and the priority
17 information, threads for the first components and for the second components, each
18 of the threads having a priority.
19

20 9. The media of claim 8, further comprising receiving usage information
21 for the first multimedia pipeline or the second multimedia pipeline, the usage
22 information indicating how media of the first or second multimedia pipelines is to
23 be used or viewed.
24
25

1 **10.** The media of claim 9, wherein the act of allocating comprises
2 assigning priority to one of more of the threads based on the usage information.
3

4 **11.** The media of claim 8, further comprising receiving value
5 information for the first multimedia pipeline or the second multimedia pipeline,
6 the value information indicating a value of media of the first or second multimedia
7 pipelines.
8

9 **12.** The media of claim 11, wherein the act of allocating comprises
10 assigning priority to one or more of the threads based on the value information.
11

12 **13.** The media of claim 8, further comprising ascertaining a first
13 topology for the first multimedia pipeline based on the input/output characteristics
14 for the first components and a second topology for the second multimedia pipeline
15 based on the input/output characteristics for the second components and wherein
16 the act of allocating comprises allocating based on the first topology and the
17 second topology.
18

19 **14.** The media of claim 8, further comprising instructing the first
20 components and the second components to refrain from opening threads.
21

22 **15.** The media of claim 8, further comprising requesting the input/output
23 information.
24
25

1 **16.** The media of claim 15, wherein the act of requesting comprises
2 requesting the input/output information from a media application comprising the
3 first components and the second components.

4
5 **17.** The media of claim 8, further comprising executing the first
6 components and the second components according to the threads and the priority
7 of the threads.

8
9 **18.** The media of claim 17, wherein at least one thread is executed on a
10 first processor and at least one thread is executed on a second processor.

1 **19.** One or more computer-readable media having computer-readable
2 instructions therein that, when executed by a computer, cause the computer to
3 perform acts comprising:

4 receiving input/output information for components of a multimedia
5 pipeline, the pipeline having a composition component having inputs from two or
6 more parallel components, the parallel components capable of being processed in
7 parallel;

8 receiving timing information for the parallel components and their input
9 components, the input components for each parallel component providing direct or
10 indirect input to each parallel component;

11 ascertaining whether performing each of the parallel components and their
12 input components in separate threads or in one thread is faster for a multi-core or
13 multi-processor computing device based on the timing information and a time
14 needed for the computing device to switch between threads; and

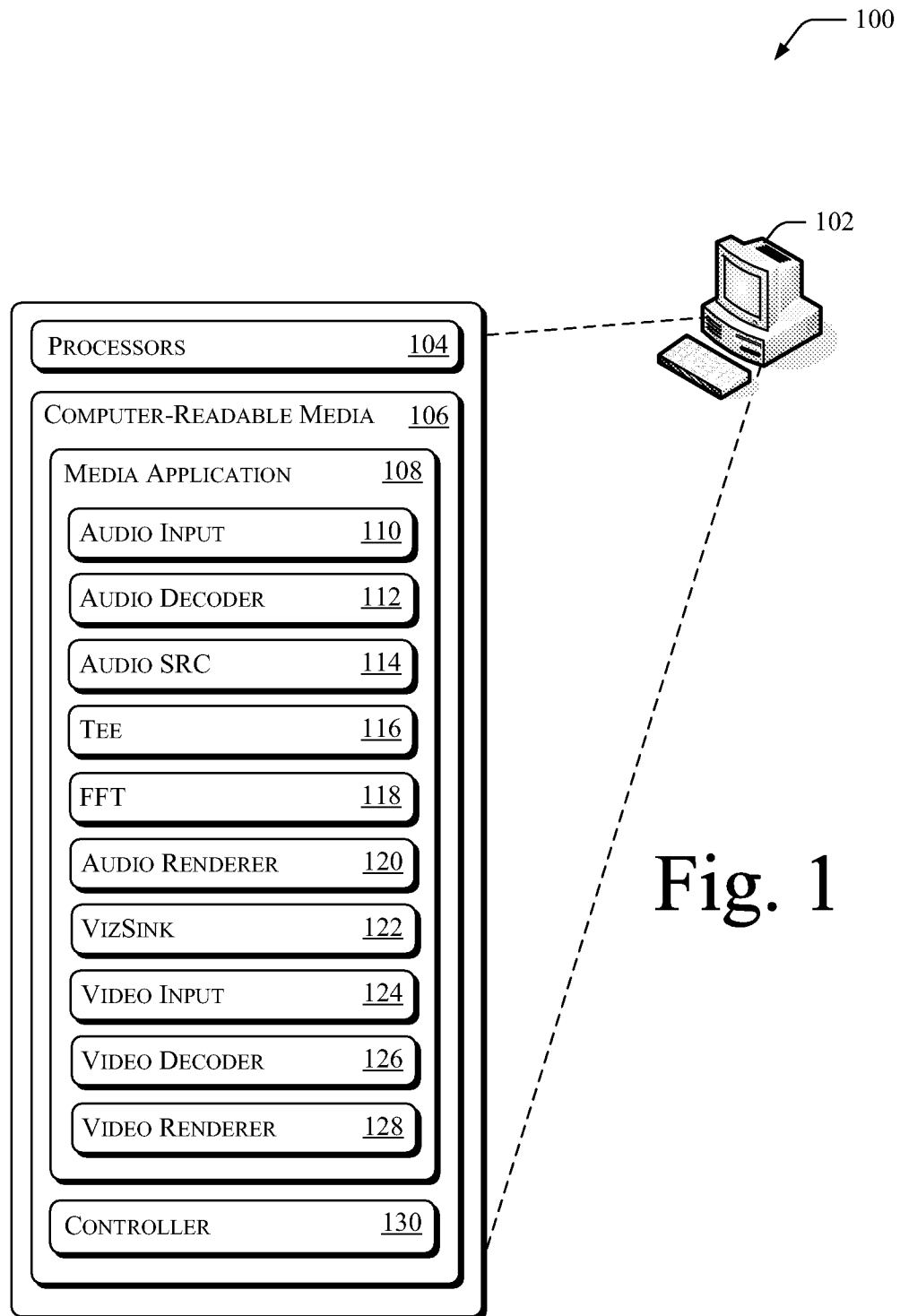
15 allocating separate threads to the parallel components and their input
16 components if performing the separate threads is faster than performing the one
17 thread.

18
19 **20.** The media of claim 19, wherein the timing information comprises
20 likely processing resource usage for each of the parallel components and their
21 input components.

1 **ABSTRACT**

2 Systems and/or methods are described that enable multi-threaded
3 multimedia processing. These systems and/or methods may, in some
4 embodiments, allocate threads for components of a multimedia pipeline based on
5 input/output characteristics of the components. These systems and/or methods
6 may also allocate threads and priorities for those threads based on a relative
7 importance given components of two or more multimedia pipelines.

8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25



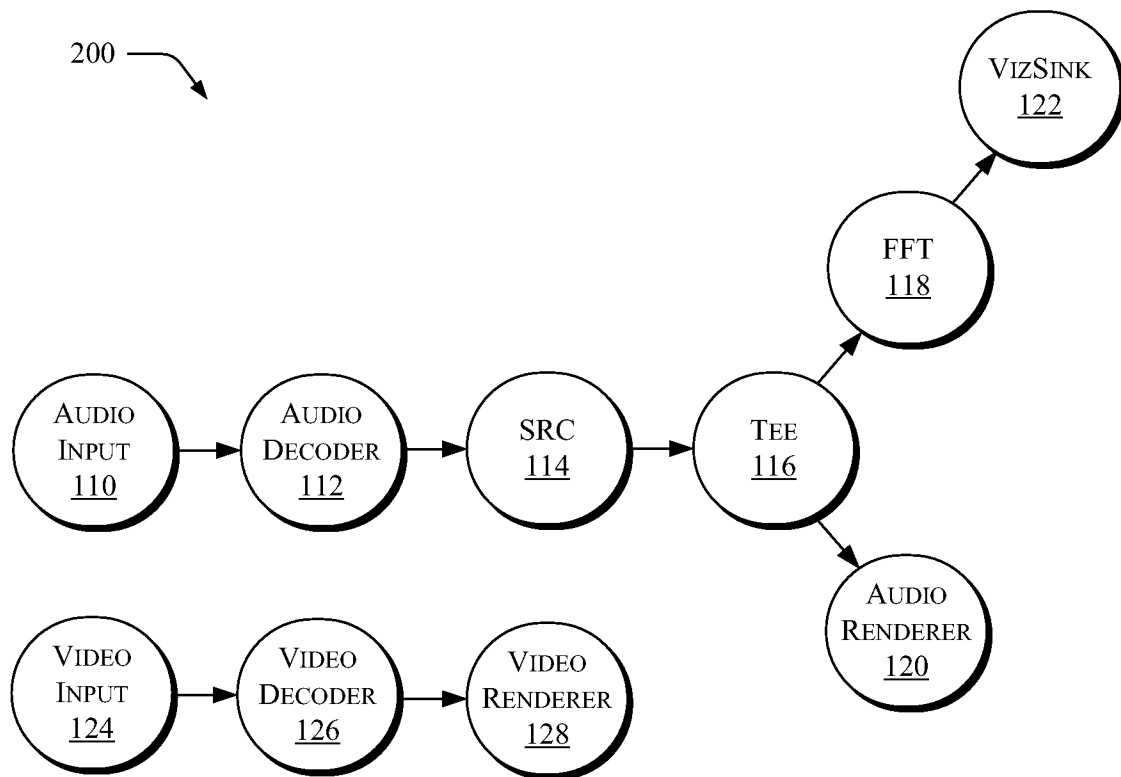


Fig. 2

lee hayes

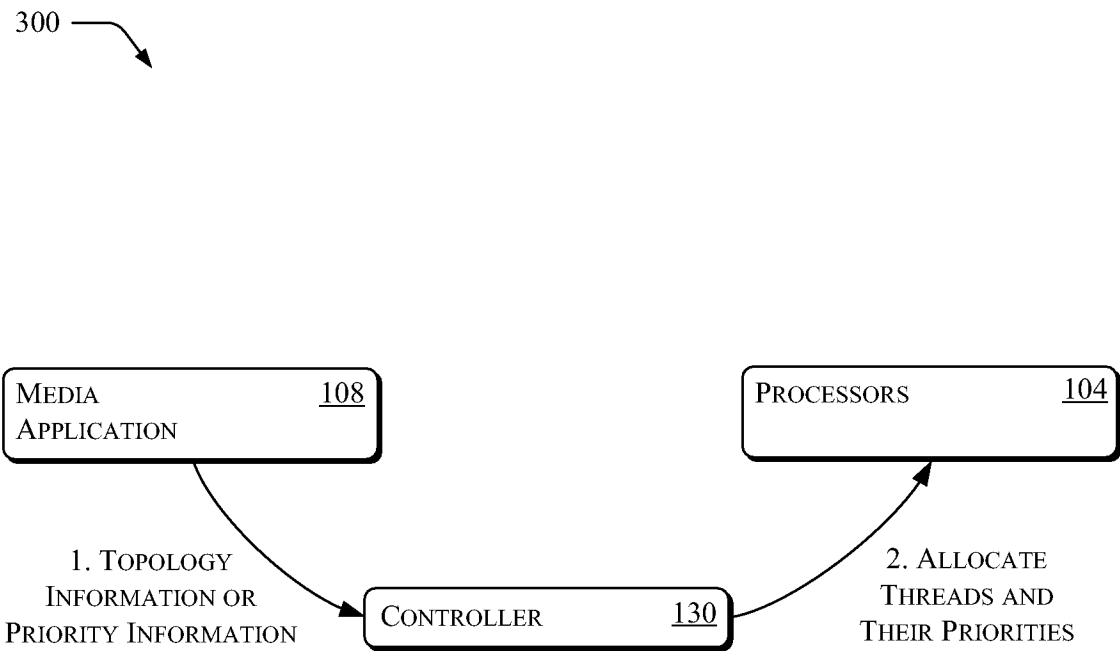


Fig. 3

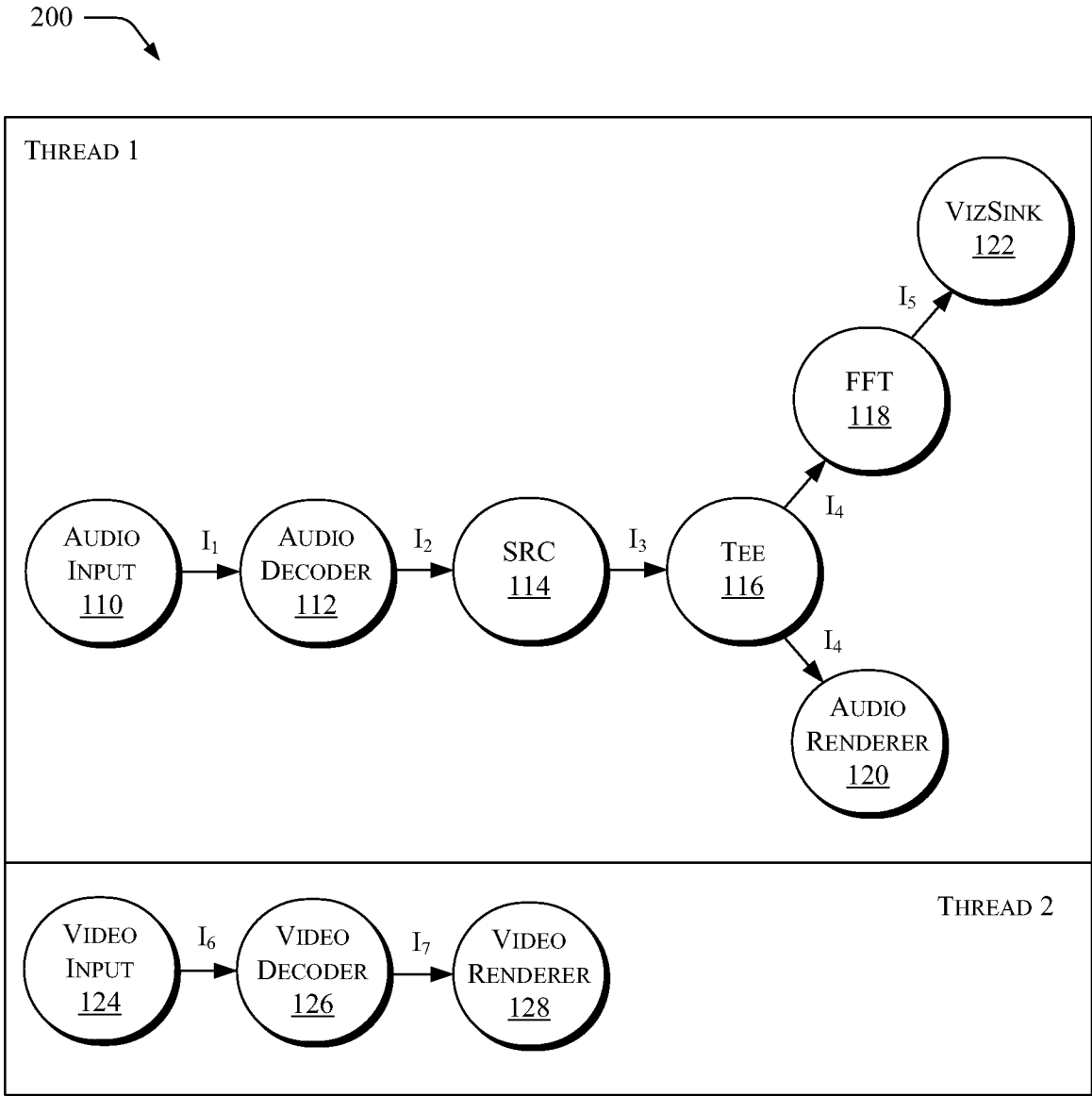


Fig. 4

lee@hayes

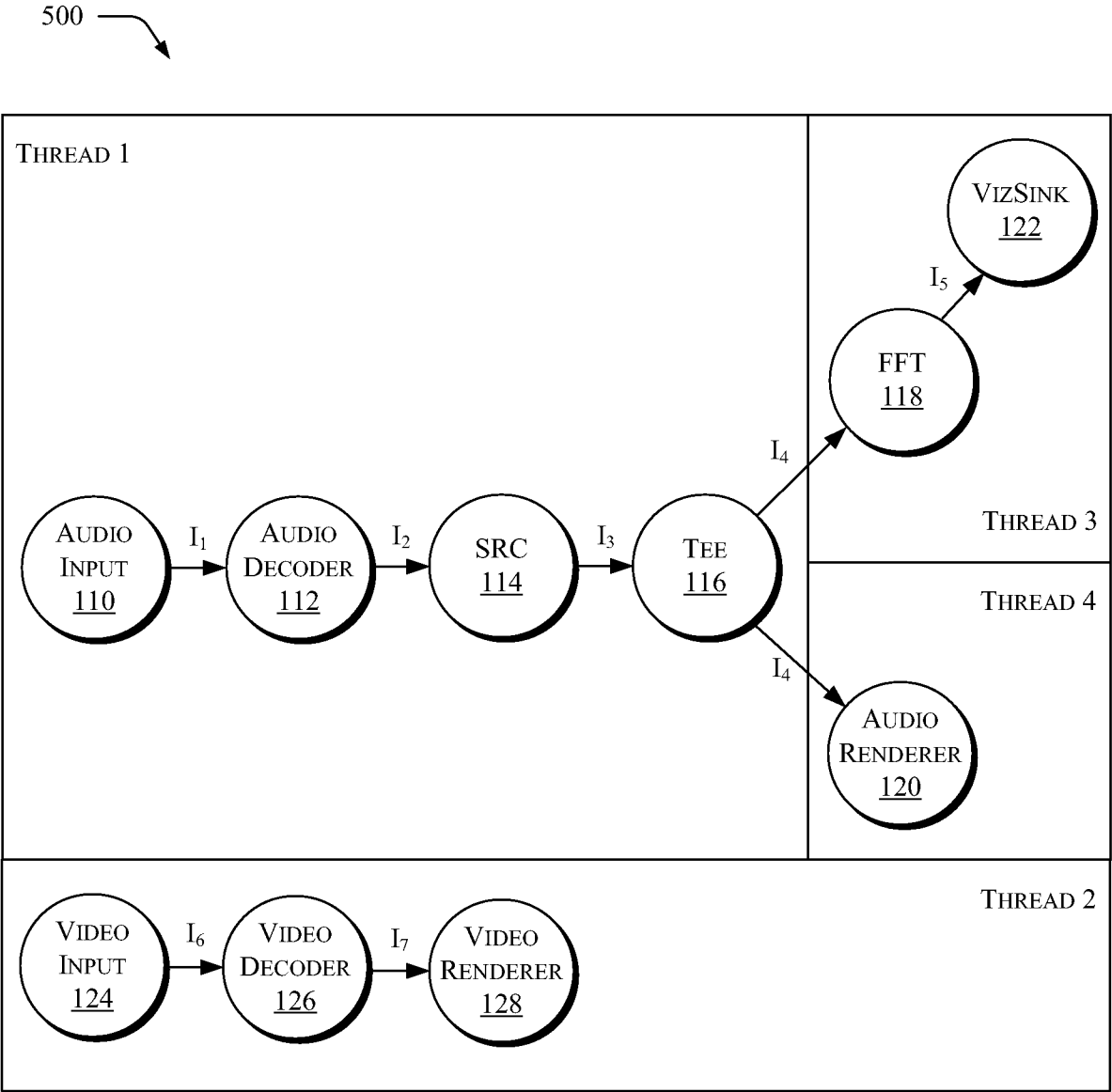


Fig. 5

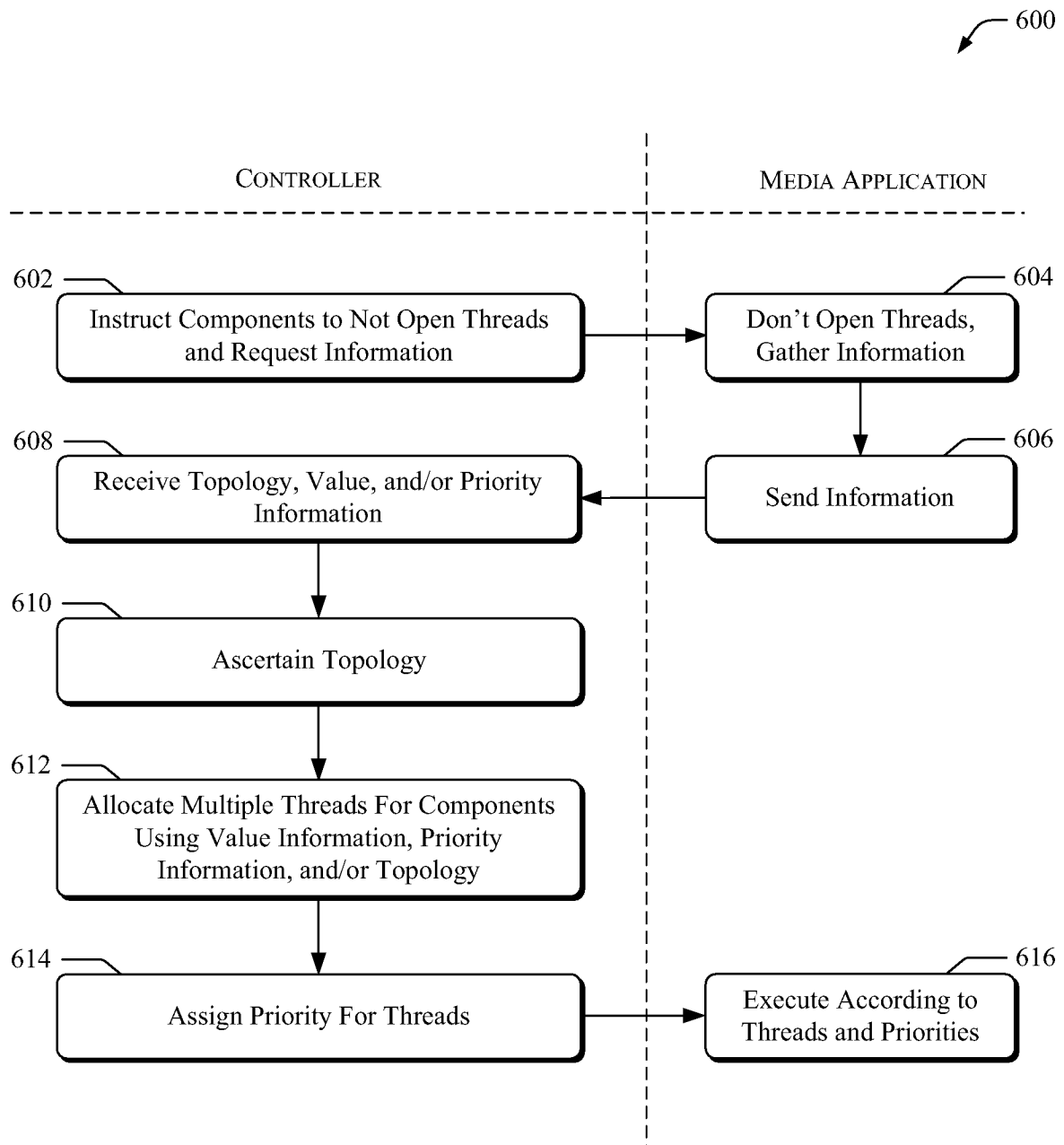


Fig. 6

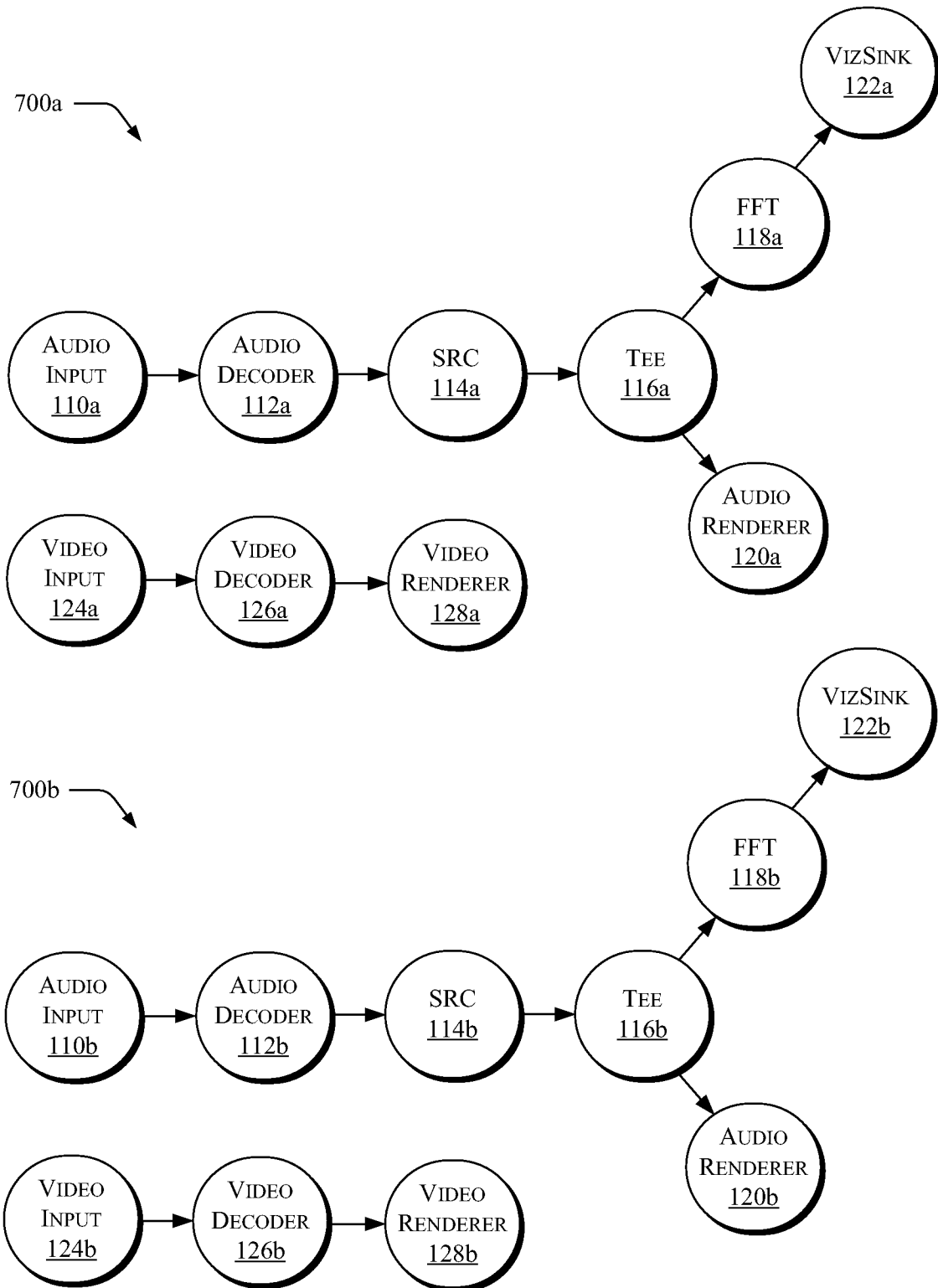


Fig. 7

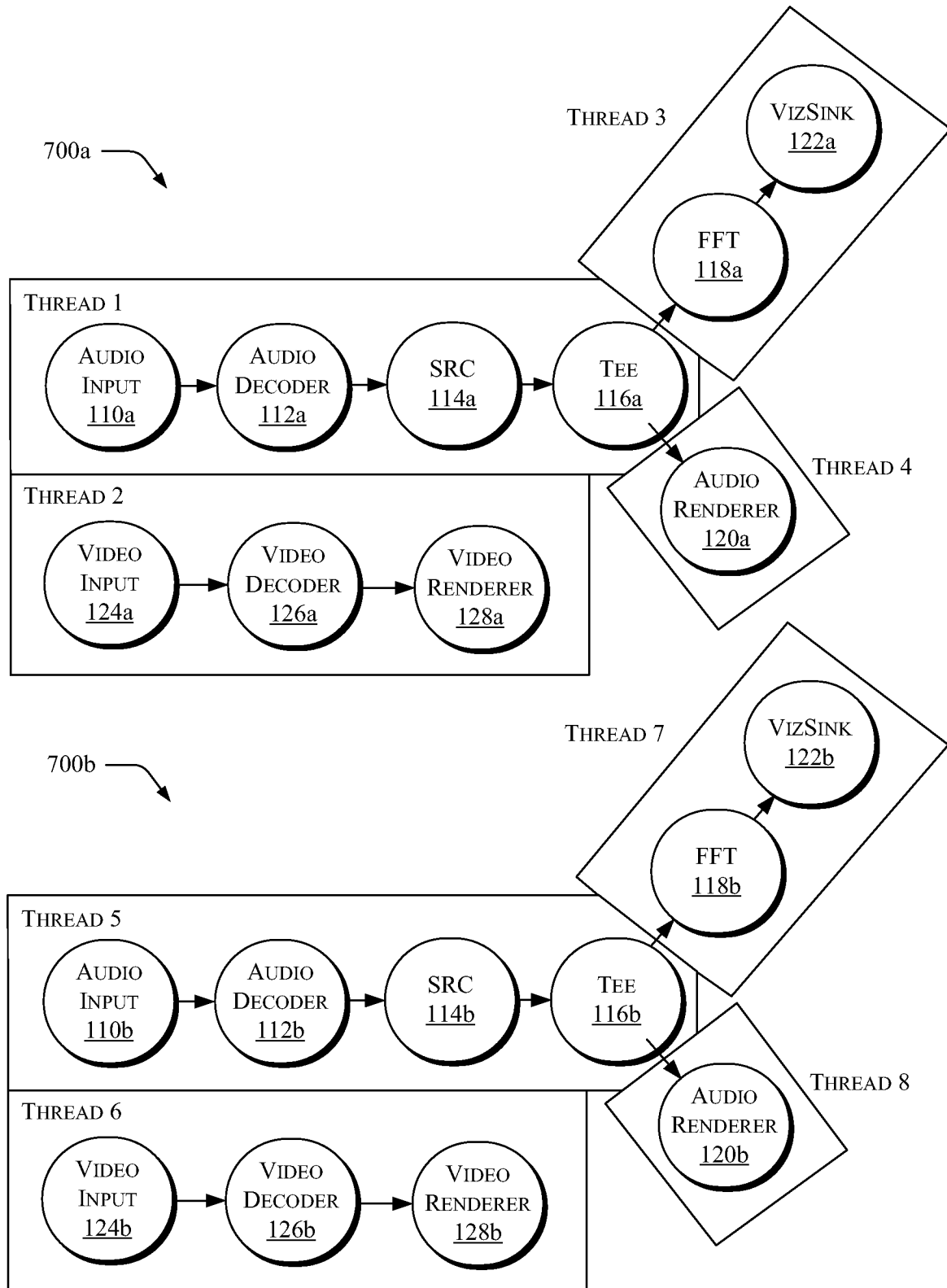


Fig. 8

lee & hayes

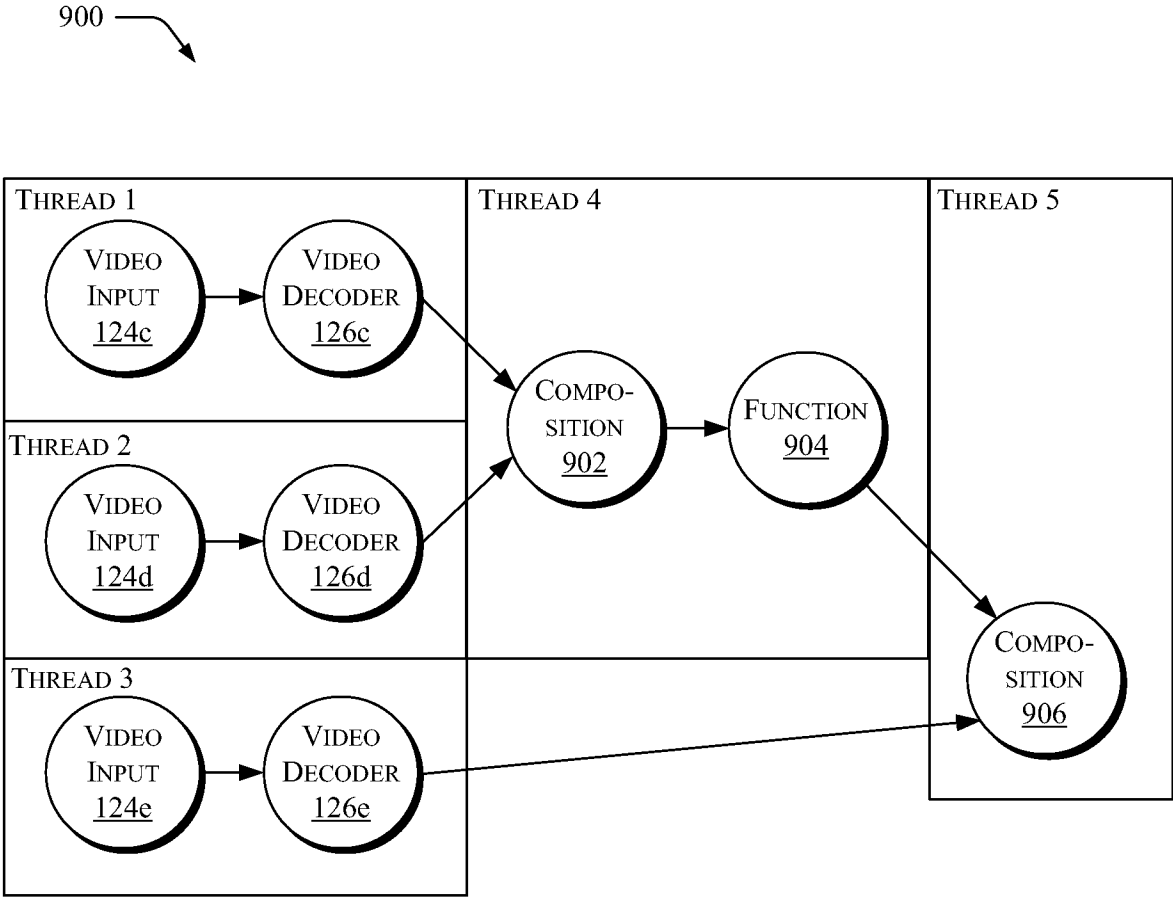


Fig. 9

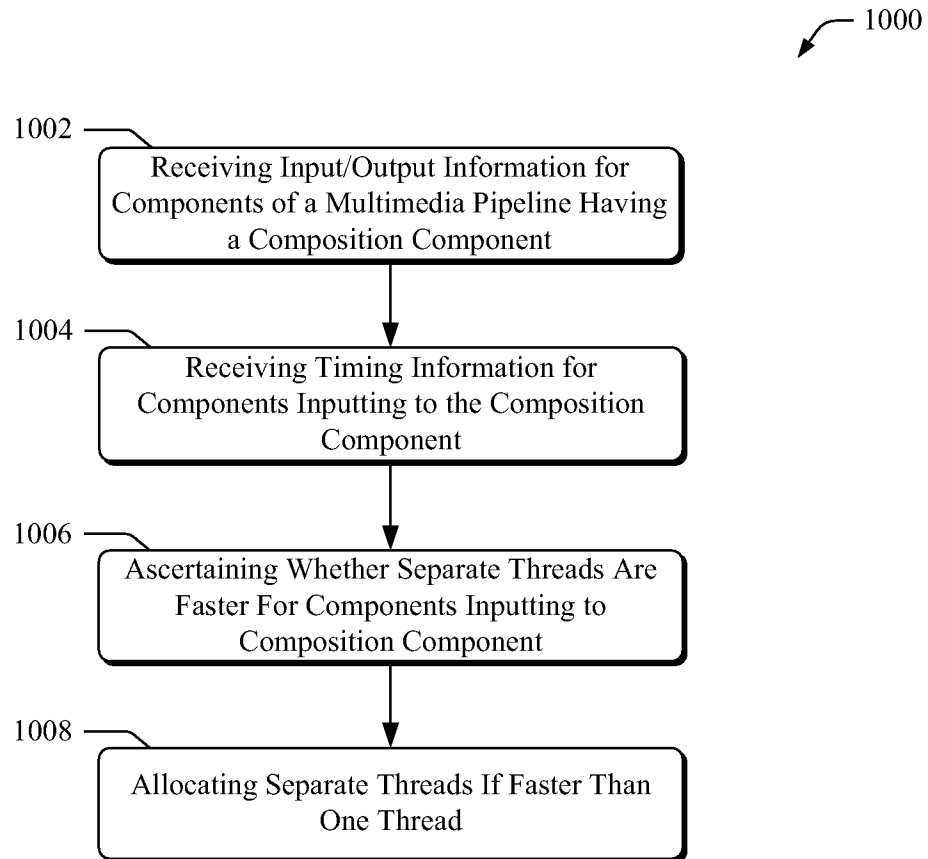


Fig. 10